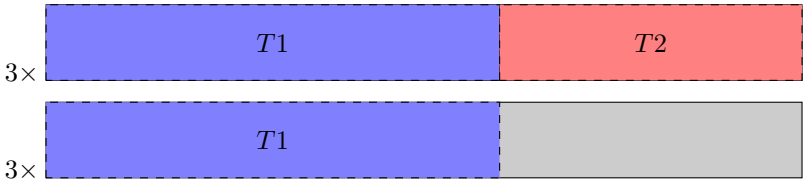
Feuille de travaux pratiques n° 3
Résolution exacte du problème de bin-packing par des méthodes simples

1 Définition du problème

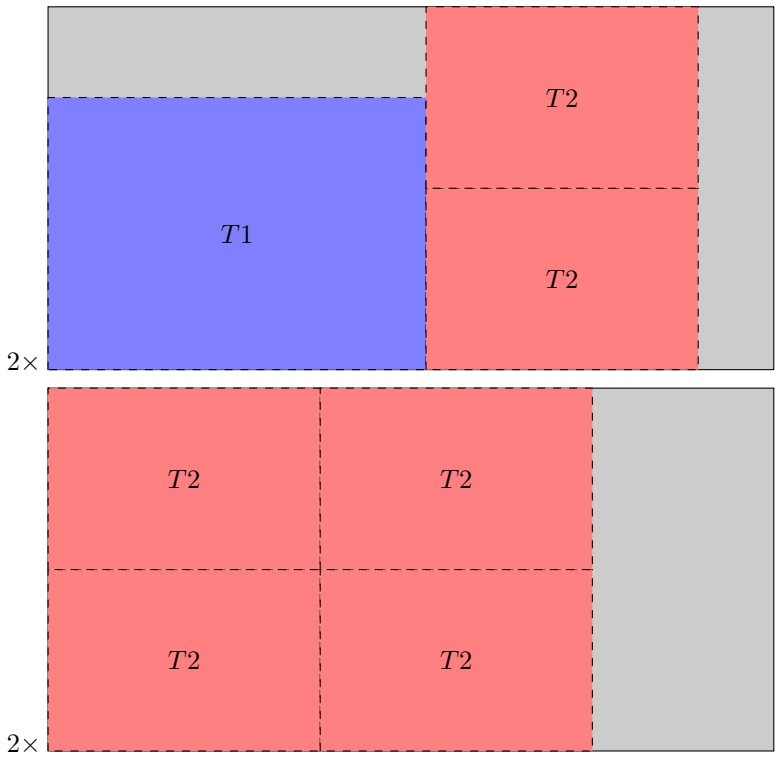
Le problème de bin-packing consiste à déterminer le nombre minimal de conteneurs (*bins* en anglais) identiques pour ranger un ensemble d'objets de tailles diverses, sans qu'ils ne se chevauchent. Ce problème a de nombreuses variantes. Nous en considèrerons ici deux parmi les plus simples : la variante *mono-dimensionnelle* et la variante *bi-dimensionnelle avec orientation*. Contrairement à de nombreux problèmes présentés en cours, il n'est pas immédiat de poser une modélisation pour ce problème. Le problème sera donc présenté dans un premier temps à l'aide d'exemples.

Nous considérons d'abord la variante mono-dimensionnelle. Nous supposons pour l'exemple que la taille (également appelée longueur) des bins est de 10. Nous souhaitons trouver le nombre minimal de bins de manière à ranger 2 objets de taille 4 (type 1) et 4 objets de taille 3 (type 2). Une solution optimale consiste à utiliser 2 bins contenant chacun 1 objet de type 1 et deux objets de type 2.

Afin de minimiser le nombre de bins à utiliser, il est naturel de chercher à remplir les bins autant que possible. Dans l'idéal, on réussit à les remplir complètement comme c'est le cas dans ce premier exemple. Cependant, cela ne sera pas possible en général. Par exemple, si on souhaite ranger 6 objets de taille 6 (type 1) et 3 objets de taille 4 (type 2), une solution optimale consiste en 3 bins composés de 1 objet de type 1 et 1 objet de type 2, et 3 bins composés de 1 objet de type 1. On a donc un total de 6 bins. Seul le premier type de bin est complètement rempli. Cela est ici dû au fait qu'il est impossible de mettre plus d'un objet de type 1 dans un bin.



Nous considérons ensuite la variante bi-dimensionnelle avec orientation. Les bins ainsi que les objets à y ranger sont maintenant rectangulaires. Ils n'ont donc pas seulement une largeur mais aussi une hauteur. Le problème consiste toujours à placer l'ensemble des objets sans chevauchement dans un nombre minimal de bins. Dans la variante que nous considérons (avec orientation), la rotation des objets n'est pas autorisée. Nous passons maintenant à un exemple. Nous supposons pour l'exemple que la taille des bins est de 48×96, et que nous souhaitons y ranger 2 objets de taille 36×50 (type 1) et 12 objets de taille 24×36 (type 2). Une solution optimale consiste en 2 bins composés de 1 objet de type 1 et 2 objets de type 2, et 2 bins composés de 4 objets de type 2. On a donc ici un total de 4 bins.



Une différence essentielle avec le cas mono-dimensionnel est que les objets ne sont pas placés simplement les uns à la suite des autres. Une notion de placement des objets dans le bin apparaît ici.

2 Méthodes de résolution pour le problème de bin-packing mono-dimensionnel

Nous considérons ci-dessous trois méthodes de résolution : une heuristique et deux méthodes de résolution exacte.

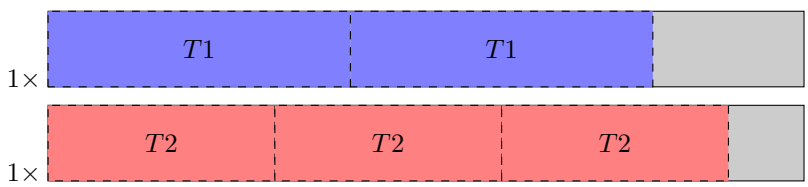
2.1 Heuristique best-fit

Le principe d'une heuristique est de construire rapidement une solution admissible sans avoir nécessairement de garantie théorique sur la qualité de la solution obtenue. Une heuristique extrêmement simple pourrait consister à mettre uniquement un objet dans chaque bin. Ainsi, nous sommes certains qu'il n'y a pas de chevauchement d'objets. Cependant, le nombre de bins est alors égal au nombre d'objets et nous sommes alors loin de minimiser le nombre de bins !

Il est assez simple de construire un algorithme glouton pour ce problème. Le principe de ce type de méthode est de faire des choix définitifs, i.e. si nous faisons le choix de mettre un objet dans un bin, nous ne revenons pas en arrière. Pour ce problème, deux questions se posent dans la définition d'un algorithme glouton. Tout d'abord, on doit choisir dans quel ordre on va insérer les objets. Ensuite, pour chaque objet, il faut décider si on l'insère dans un bin existant (si possible), ou alors si on ouvre un nouveau bin pour l'insérer. Dans le cas de l'heuristique *best-fit*. Les choix suivants sont effectuées.

1. On trie les objets par taille décroissante,
2. On range chaque objet (dans l'ordre du tri) dans le bin ouvert le plus rempli, parmi les bins pouvant l'accueillir. Si aucun bin ouvert n'a la place suffisante, on ouvre un nouveau bin.

Une application au problème de l'exemple de la section 1 donne la solution suivante.



Comme on le voit, il ne s'agit pas d'une solution optimale (que nous ne sommes pas censés connaître en général). Sur cet exemple, nous obtenons tout de même l'information qu'une solution optimale nécessite au plus 3 bins, autrement dit que 3 est une borne supérieure sur le nombre optimal de bins.

2.2 Modélisation directe

Il existe une modélisation directe pour le problème de bin-packing mono-dimensionnel. Ce modèle fonctionne avec deux ensembles de variables de décision.

$$x_{ij} = \begin{cases} 1 \text{ si l'objet } i \text{ est rangé dans le bin } j \\ 0 \text{ sinon} \end{cases}$$

$$y_j = \begin{cases} 1 \text{ s'il y a un objet rangé dans le bin } j \\ 0 \text{ sinon} \end{cases}$$

où $i \in \{1, \ldots, n\}$ avec n le nombre d'objets (pas le nombre de type d'objets) à ranger dans les bins, et $j \in \{1, \ldots, m\}$ avec m un nombre de bins supérieur ou égal au nombre optimal de bins. Pour $i \in \{1, \ldots, n\}$, t_i est une donnée représentant la taille de l'objet i, et T représente la taille d'un bin. On obtient ensuite la modélisation suivante.

$$\begin{array}{rlll} \min\ z & = & \sum_{j=1}^{m} y_j & \\ s.c. & \sum_{j=1}^{m} x_{ij} & = \quad 1, & i \in \{1, \ldots, n\} \\ & \sum_{i=1}^{n} t_i x_{ij} & \leq \quad T y_j, & j \in \{1, \ldots, m\} \\ & x_{ij} \in \{0,1\}, & & i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, \\ & y_j \in \{0,1\}, & & j \in \{1, \ldots, m\} \end{array}$$

La fonction objectif compte le nombre de bins ouverts, la première contrainte oblige chaque objet à être rangé dans un bin. La deuxième contrainte spécifie que la somme des tailles des objets rangés dans un bin ne peut être supérieure à la taille du bin, qui devient nulle si le bin n'est pas ouvert (ne contient aucun objet).

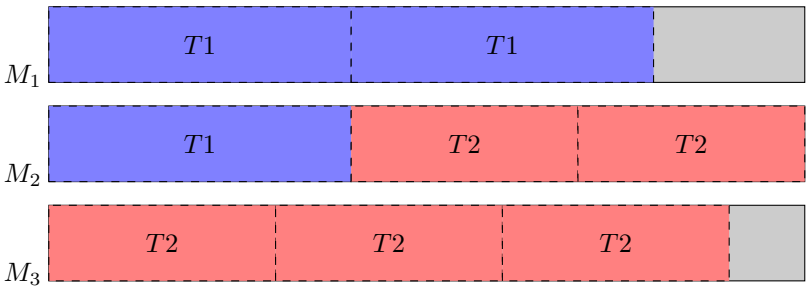
Afin d'utiliser cette modélisation, il faut définir une valeur raisonnable pour m. En effet, si m est strictement inférieur au nombre optimal de bins, on se retrouve avec un problème impossible. Si on prend une valeur "large" pour m, on obtient alors une modélisation avec un nombre excessivement important de variables. Il faut donc juste connaître une (bonne) borne supérieure sur le nombre optimal de bins. L'heuristique best-fit présentée dans la section 2.1 peut donc être utilisée pour fixer la valeur de m.

Remarque : comme indiqué dès la définition des variables de décision, la possibilité que des objets soient de même taille n'est pas exploitée. Pourtant, c'est une caractéristique réaliste pour des instances numériques de ce problème. Il sera attendu un modèle direct alternatif pour remédier à cela. Ce modèle ne comportera d'ailleurs pas nécessairement que des variables binaires.

2.3 Modélisation indirecte

Avant de poser une modélisation, nous allons ici d'abord effectuer un traitement important sur les données. Clairement, on va devoir remplir les bins autant que possible pour obtenir une solution optimale. Nous allons donc déterminer les différentes possibilités de remplissage maximum des bins. Nous appellerons *motifs* les bins ainsi remplis.

Dans le cas de l'exemple, il y a 3 motifs possibles.

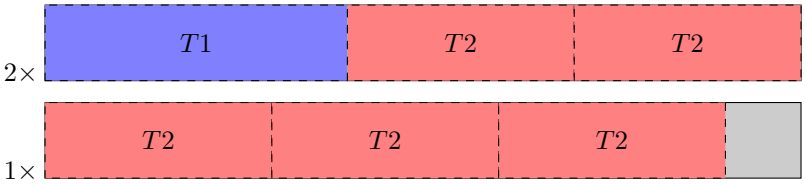


Connaissant ces motifs, nous pouvons poser une modélisation en associant une variable $x_i \in \mathbb{N}$ à chaque motif M_i et représentant le nombre de bins correspondant au motif i. Connaissant le nombre d'objets de chaque type à ranger dans les bins, et le nombre d'objets de chaque type dans chaque motif, nous pouvons donc écrire la modélisation suivante.

$$\begin{array}{lrrrrrr} \min z & = & x_1 & +x_2 & +x_3 & & \\ & s.c. & 2x_1 & +x_2 & & \geq & 2 \\ & & & +2x_2 & +3x_3 & \geq & 4 \\ & & x_1, & x_2 & \in & \mathbb{N} & \end{array}$$

La fonction objectif indique le nombre de bins utilisés. À chaque type d'objet est associée une contrainte. Le membre de gauche spécifie le nombre d'objets de ce type qui sont placés dans les bins, et ce nombre doit être supérieur ou égal au nombre d'objets de ce type à placer.

En résolvant ce programme linéaire en variables entières, on tombe directement sur la solution optimale attendue. Cependant, il est possible qu'on ait trop d'objets tout en ayant le bon nombre de bins. Si on avait 5 objets de type 2 à placer au lieu de 4. Une solution optimale pouvant être obtenue par la résolution du programme linéaire en variables entières serait la suivante.



On a donc 7 objets de type 2 au lieu de 5. Pour obtenir une solution optimale correcte, il suffit de supprimer 2 objets de taille 2 arbitrairement sur l'ensemble des bins. Ce post-traitement est immédiat à appliquer.

Plus généralement, pour un problème avec m types d'objets différents, dans lequel n motifs différents ont été identifiés, la modélisation s'écrit de la façon suivante.

$$\begin{array}{lrlll} \min z & = & \displaystyle\sum_{j=1}^{n} x_j & & \\ & s.c. & \displaystyle\sum_{j=1}^{n} a_{ij} x_j & \geq & d_i, \quad i \in \{1, \ldots, m\} \\ & & x_j \in \mathbb{N}, & & j \in \{1, \ldots, n\} \end{array}$$

où d_i représente le nombre d'objets de type i à ranger dans les bins, et a_{ij} représente le nombre d'objets de type i dans le motif j.

Il reste maintenant à préciser comment effectuer cette énumération des motifs. Une manière simple de réaliser cela est appliquée ci-dessous dans le cas de l'exemple.

On met un objet de type 1 dans un bin.

On ajoute un second objet de type 1.

Ensuite, nous essayons d'ajouter un autre objet de type 1 mais cela est impossible. Nous essayons alors d'ajouter un objet de type 2 et cela est également impossible. Nous n'avons plus aucun type d'objet à essayer, ce bin est donc rempli autant que possible. Nous avons donc identifié un premier motif (M_1). Pour pouvoir à nouveau ajouter des objets, nous devons en retirer un. Nous retirons donc le dernier objet de type 1 qui a été placé.

Nous pouvons maintenant ajouter un objet de type 2 (nous avons déjà traité le cas de l'ajout d'un objet de type 1, nous passons donc à la suite).

Nous ajoutons ensuite un autre objet de type 2.

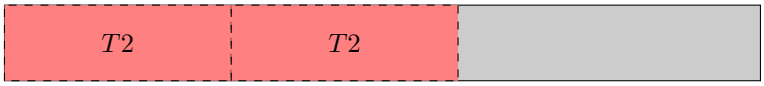
Il est ensuite impossible d'ajouter un autre objet de type 2 et il n'y a aucun autre type d'objet à essayer d'ajouter. Nous avons identifié un autre motif (M_2). Pour continuer, nous retirons un objet de type 2,

Comme nous n'avons aucun autre type d'objet à essayer d'ajouter à la place de l'objet de type 2 que nous venons de retirer, nous retirons un autre objet de type 2 pour continuer,

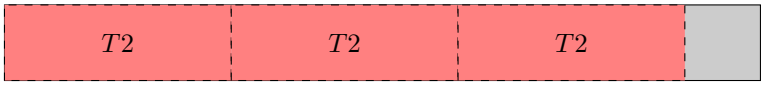
De même, nous retirons le premier objet de type 1 ayant été inséré (toutes les combinaisons possibles avec un objet de type 1 ont été énumérées).

Nous ajoutons un objet de type 2

Puis un autre (pourquoi considérer l'ajout d'un objet de type 1 est ici inutile ?),



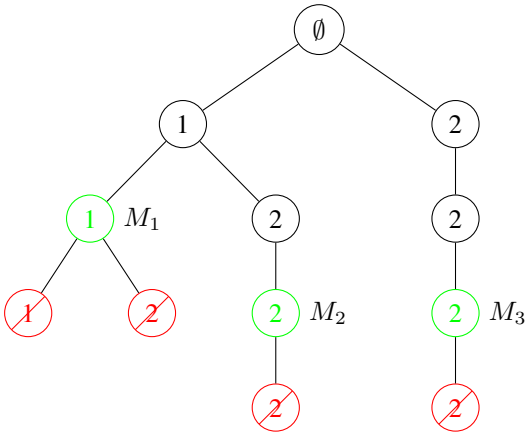
Puis un autre.



Il est finalement impossible d'ajouter un autre objet de type 2 et il n'y à aucun autre type d'objet à ajouter. Nous avons donc identifié un autre motif ($M3$). Nous vidons ensuite le bin dans les étapes suivantes sans avoir la possibilité d'effectuer des ajouts supplémentaires.

L'énumération effectuée peut être résumée de manière arborescente. Cela n'implique pas qu'une structure d'arbre est nécessaire pour réaliser cette énumération des motifs, mais qu'il sera naturel d'écrire une fonction

récursive pour cela. Il restera à écrire formellement cette fonction indépendamment du nombre de types d'objet et de la taille des bins.



3 Méthode de résolution pour le problème de bin-packing bi-dimensionnel avec orientation

Nous ne considérerons qu'une seule méthode dans le cas bi-dimensionnel et cette méthode sera très similaire à celle présentée dans la section 2.3. De la même façon que dans le cas mono-dimensionnel, il est naturel de chercher à remplir les bins au maximum. Nous avons ainsi la même notion de motifs que dans le cas mono-dimensionnel.

Dans le cas de l'exemple introduit en fin de section 1, il y a seulement deux motifs avec les compositions suivantes :

— 1 objet de type 1 et 2 objet de type 2
— 4 objets de type 2

Une fois ces motifs connus, le programme linéaire en variables entières à résoudre se construit à l'identique du cas mono-dimensionnel.

De plus, l'énumération des motifs peut se faire par la même méthode que dans le cas mono-dimensionnel à une exception près : lorsqu'on souhaite savoir s'il est possible d'ajouter un objet dans un bin, la notion de placement des objets intervient. Pour apporter une réponse à cette question, une approche simple consiste à poser un nouveau programme linéaire.

Nous commençons par le cas basique du placement de deux objets dans un bin (le cas d'un seul objet ne pose pas de question). Nous cherchons donc à savoir s'il est possible de placer un objet i de taille $l_i \times h_i$ et un objet j de taille $l_j \times h_j$ dans un bin de taille $L \times H$ (il est possible que $l_i = l_j$ et $h_i = h_j$). Nous associons respectivement les variables continues $x_i \geq 0$ et $y_i \geq 0$, ainsi que $x_j \geq 0$ et $y_j \geq 0$, aux objets i et j. Ces variables indiquent la position du point inférieur-gauche de l'objet dans le bin. On a donc les contraintes suivantes indiquant qu'aucun des objets ne déborde du bin.

$$x_i + l_i \leq L$$
$$y_i + h_i \leq H$$
$$x_j + l_j \leq L$$
$$y_j + h_j \leq H$$

On peut remarquer que ces contraintes sont en realité de simple bornes sur chacune des variables.

Il faut ensuite s'assurer qu'un placement sans chevauchement de ces deux objets est possible dans le bin. Il n'y aura pas de chevauchement si au moins une des conditions suivantes est vérifiée :

— l'objet i est à gauche de l'objet j : $x_i + l_i \leq x_j$,
— l'objet i est à droite de l'objet j : $x_j + l_j \leq x_i$,
— l'objet i est en-dessous de l'objet j : $y_i + h_i \leq y_j$,
— l'objet i est au-dessus de l'objet j : $y_j + h_j \leq y_i$.

Il sera donc nécessaire de satisfaire au moins une des contraintes ci-dessus, ce que nous pouvons formuler en ajoutant 4 variables binaires $b_1, b_2, b_3, b_4 \in \{0, 1\}$ et une contrainte additionnelle. Nous obtenons donc

$$x_i + l_i \leq x_j + M(1 - b_1)$$

$$x_j + l_j \leq x_i + M(1 - b_2)$$
$$y_i + h_i \leq y_j + M(1 - b_3)$$
$$y_j + h_j \leq y_i + M(1 - b_4)$$
$$b_1 + b_2 + b_3 + b_4 \geq 1$$

où M est un nombre suffisamment large pour lequel on pourra montrer que L ou H (suivant les contraintes) est une valeur convenable.

On peut remarquer qu'il n'y a pas de fonction objectif ici. Tout ce que nous souhaitons savoir, c'est s'il existe une solution admissible au problème ainsi posé. Nous parlons donc ici de problème de *faisabilité* et pas de problème d'optimisation car il n'y a pas de fonction objectif dans ce problème (de manière équivalente, on peut considérer qu'on a une fonction objectif constante nulle). La résolution de ce type de problème permet ici de trouver un placement approprié des objets à placer si cela est possible, ou d'obtenir l'information qu'aucun placement n'est possible.

Passons maintenant au cas général, on souhaite savoir si on peut insérer simultanément l'ensemble des objets $\{i_1, i_2, \ldots, i_r\}$ de taille $l_k \times h_k$ pour $k \in \{1, \ldots, r\}$, dans un bin de taille $L \times H$. Nous considérons donc les variables continues $x_k \geq 0$ et $y_k \geq 0$ correspondant à la position du point inférieur gauche de l'objet k pour $k \in \{1, \ldots, r\}$. Chacun de ces objets devra tenir dans le bin, nous aurons donc les contraintes (bornes) suivantes.

$$x_k + l_k \leq L, k \in \{1, \ldots, r\}$$

$$y_k + h_k \leq H, k \in \{1, \ldots, r\}$$

Ensuite, il faut spécifier qu'il n'y a pas de chevauchement entre les objets. Cela demande de spécifier qu'il n'y a pas de chevauchement entre les paires d'objets. Pour chaque paire d'objets (i, j) avec $i < j$ et $i, j \in \{1, \ldots, r\}$, on aura donc les 4 variables binaires $b_{(i,j),1}, b_{(i,j),2}, b_{(i,j),3}, b_{(i,j),4} \in \{0, 1\}$ et les contraintes suivantes.

$$x_i + l_i \leq x_j + M(1 - b_{(i,j),1})$$

$$x_j + l_j \leq x_i + M(1 - b_{(i,j),2})$$
$$y_i + h_i \leq y_j + M(1 - b_{(i,j),3})$$
$$y_j + h_j \leq y_i + M(1 - b_{(i,j),4})$$
$$b_{(i,j),1} + b_{(i,j),2} + b_{(i,j),3} + b_{(i,j),4} \geq 1$$

Les variables binaires ajoutées sont différentes pour chaque paire de variables, d'où le choix ici effectué dans leur nommage. Cela implique également que le nombre de variables binaires ajoutées au modèle ainsi que le nombre de contraintes s'accroissent très rapidement avec le nombre d'objets qu'on essaie d'insérer dans un bin.

4 Travail à effectuer

Une implémentation des méthodes présentées dans les sections 2.1, 2.2, 2.3 et 3 est attendue en utilisant Julia/-JuMP/GLPK.

Plusieurs fichiers sont disponibles sur madoc dans l'archive `Projet 2022.zip` :
— Le fichier `Projet_NOM1_NOM2.jl` contenant un squelette à compléter,
— Un dossier `Instances` contenant deux sous-dossiers `1Dim` et `2Dim`, contenant chacun des séries d'instances numériques à résoudre.

Les instances numériques sont des fichiers textes dont le format est donné par :
— La première ligne indique la taille du bin et le nombre de types de pièce,
— Les lignes suivantes indiquent les tailles des pièces pour chaque type, ainsi que leur nombre.

Le fichier des données de l'exemple utilisé pour présenter le problème de sac à dos mono-dimensionnel est indiqué ci-dessous.

```
10 2
4 2
3 4
```

Le fichier des données de l'exemple utilisé pour présenter le problème de sac à dos bi-dimensionnel avec orientation est indiqué ci-dessous.

```
48 96 2
36 50 2
24 36 12
```

La date limite de remise des projets est fixée par défaut au dimanche 3 avril à 23h59. Le(s) code(s) source(s) ainsi que le rapport au format `.pdf` devront être remis sur madoc dans une archive (au format `.zip` ou `.tar.gz`). Voici quelques points qui devront impérativement apparaître dans votre rapport.

1. Présenter et justifier vos choix de structures de données. Les structures de données utilisées peuvent bien entendu être différentes pour les différents algorithmes.
2. Écrire le modèle demandé à la fin de la section 2.1.
3. Poser clairement à l'aide d'un pseudo-code la fonction utilisée pour énumérer les motifs.
4. Effectuer une analyse synthétique des résultats issus de la résolution des instances numériques par vos implémentations. Cela ne signifie bien entendu pas que des scans d'écran des sorties de vos algorithmes sont attendus ! Ce qui est important est d'étudier l'évolution des temps de calcul en fonction des tailles des instances numériques pour chacune des séries d'instances, de comprendre/expliquer d'où viennent les différences. Dans le cas des méthodes des sections 2.3 et 3, il sera par exemple intéressant d'étudier l'évolution du nombre de motifs. Enfin, la comparaison entre les différentes méthodes implémentées dans le cas mono-dimensionnel sera bien sûr attendue.
5. (Optionnel/Bonus) Des propositions d'amélioration (implémentées ou non) peuvent aussi faire partie du rapport.

5 Annexe

Ce projet ne nécessite pas une exploration importante de la documentation du langage Julia. Le projet a été préparé en utilisant uniquement des fonctions déjà utilisées dans les TP précédents ou présentées dans le support de cours. Les seules exceptions sont les fonctions suivantes : $sort!$, $deepcopy$, set_upper_bound. En tapant un point d'interrogation dans le REPL, on peut avoir de l'aide sur ces fonctions.

Il est bien sûr possible de demander à google de l'aide lorsqu'on se pose une question. Cependant, l'expérience passée a montré que cela a le plus souvent mené des étudiants à utiliser des bibliothèques de fonctions inadaptées pour leur besoin, avec pour conséquence un code complexe à réaliser, complexe à lire et finalement lent à l'exécution. Avant de chercher à utiliser une bibliothèque de fonctions externe, il faudra s'assurer de sa pertinence et ne pas exclure des solutions simples.